SPECIFICATION

Electronic Version 1.2.8

Stylesheet Version 1.0

Method And System For Data Element Change Across Multiple Instances Of Data Base Cache

Background of Invention

[0001] 1. *Field of the Invention* The invention pertains to the field of data base cache, and more particularly to cache management with data updates.

[0002] 2. *Background of the Related Art*

[0003] Various types of database cache are known, with techniques for update or refresh of the cache upon a change in one of the data elements. Examples are J2EE Container Managed Persistence and products such as Persistence. However, these products do not provide a locking strategy that works at a logical transaction level. Most Container Managed Persistence mechanisms rely on network connectivity between the nodes to ensure cache integrity. Further, existing products are designed to work within a single cluster.

[0004] What is needed is a system and method for cache management across heterogeneous clusters.

[0005] The preceding description is not to be construed as an admission that any of the description is prior art to the present invention.

Summary of Invention

[0006] In one aspect, the invention provides a method and system for management of data cache. A database lock of a named cache is requested, the named cache is locked, a local lock of the named cache at a local node is acquired, a timestamp corresponding to the local lock is generated, a cache item of the named cache is

invalidated in the local node, the local lock of the named cache is released, a message is sent to a remote node identifying the cache item of the named cache, an acknowledgment of the message is received from the remote node, an update of the cache item of the named cache is sent, the named cache is updated and the database lock of the named cache is released.

[0007] In another aspect, the invention provides a method and system for management of data cache. A cache miss of a cache item is identified, a read lock of a named cache is requested, the named cache including the cache item, the named cache is read locked, the cache item is requested from a master locking database, the cache item is received, and the read lock of the named cache is released.

[0008] In another aspect, the invention provides a method and system for management of data cache. A cache miss of a cache item is identified, a read lock of a global database is requested, the global database including the cache item, the global database is read locked, the cache item is requested from a master locking database, the cache item is received, and the read lock of the global database is released.

[0009] In another aspect, the invention provides a method and system for management of data cache. A predetermined event is determined to have occurred, a read lock of a named cache is requested, a timestamp is requested, an indication of a read lock of the named cache is received, a timestamp is received, the received timestamp is compared with a previous timestamp, responsive to the comparison, a predetermined action is performed, and the read lock of the named cache is released.

[0010] The specific aspects and advantages of the invention described and illustrated herein are illustrative of those which can be achieved by the present invention and are not intended to be exhaustive or limiting of the possible advantages that can be realized. Thus, the aspects and advantages of this invention will be apparent from the description herein or can be learned from practicing the invention, both as embodied herein or as modified in view of any variations which may be apparent to those skilled in the art. Accordingly, the present invention resides in the novel parts, constructions, arrangements, combinations and improvements herein shown and described

Brief Description of the Drawings

[0011] The foregoing features and other aspects of the invention are explained in the following description taken in conjunction with the accompanying figures wherein:

[0012] FIG. 1 illustrates a system according to one embodiment of the invention;

[0013] FIG. 2 illustrates steps in a method of one embodiment of the invention;

[0014] FIG. 3 illustrates steps in a method of one embodiment of the invention;

[0015] FIG. 4 illustrates steps in a method of one embodiment of the invention;

[0016] FIG. 5 illustrates steps in a method of one embodiment of the invention;

[0017] FIG. 6 illustrates steps in a method of one embodiment of the invention; and

[0018] FIG. 7 illustrates steps in a method of one embodiment of the invention.

[0019] It is understood that the drawings are for illustration only and are not limiting.

Detailed Description of the Drawings

[0020] The invention solves a problem of maintaining cache coherency in a distributed system with a central database server. The cache coherency mechanism ensures that updates are synchronous across all instances of the cache in the distributed system (*i.e.* , that an update takes effect on all nodes in the system at the same time and no node is ever out of synchronization with another node.) In this way, the system is never in an inconsistent state. To accomplish this, the invention uses an integrated lock across threads, across Virtual Machines and across address spaces.

[0021] The invention has backup mechanisms in case of network failures and also allows the system to be brought back up into a consistent state after a complete failure. The invention allows caches to be taken down and brought up in different address spaces at different times but does not dictate the way that persistence is implemented (i.e., this mechanism can be used on top of existing code, allowing custom cache flushing mechanisms).

[0022] Use of a master database in one embodiment of the invention provides a way to enforce synchronization across the system using transactionality. Other known invalidation mechanisms usually either implement a custom invalidation mechanism

(such as a send/ack invalidation mechanism), or use some form of global lock. The present invention is generally easier than implementing a custom global cache synchronization mechanism. The cache implementation of the invention also provides an easy way to integrate the distributed cache with other transactional database updates. This design maps cache invalidations against logical data groupings that are different than the physical table structure of the database. The synchronization mechanism also avoids causality race conditions (sequencing, etc).

[0023] Referring first to FIG. 1, system 100 of the invention includes local node 102, cache manager 104 and remote node(s) 106. One embodiment also includes a primary or master locking database 110. Network 108 interconnects local node 102, cache manager 104 and remote node(s) 106. Where system 100 includes primary database 110, database 110 is also interconnected to the other elements of system 100 by network 108. Network 108 is any of a number of different types of wired and wireless networks, including local area network (LAN) and wide area network (WAN).

[0024] Although not illustrated in FIG. 1, each of local node 102, cache manager 104, remote node(s) 106 and database 110 include one or more fixed and removable software code storage media, processor and memory to run the software code as well as input/output devices and network interface devices.

[0025] Steps in one embodiment of the invention are illustrated in FIG. 2. At step 202, local node 102 determines that a cache element in a named cache requires modification. The need for modification may include the need for an update, or a data change.

[0026] At step 204, local node 102 requests a database lock of the named cache. It is also possible that the request for a database lock at step 204 is a request for a global lock of the master database.

[0027] At step 206, cache manager 104 receives the request for a database lock of the named cache (or global lock of the master database), and locks the named cache (or master database).

[0028] At step 208, cache manager 104 sends an indication to local node 102 that the named cache (or master database) is locked.

[0029] At step 210, local node 102 receives the indication from cache manager 104 that the named cache (or master database) is locked.

[0030] At step 212, local node 102 acquires a local write lock of the named cache. Although called a write lock, the write lock includes a read lock of the named cache.

[0031] At step 214, local node 102 sends a timestamp to cache manager 104 for use in an update of the lock table.

[0032] At step 216, cache manager 104 receives the timestamp, and at step 218 cache manager 104 updates the lock table with the timestamp from local node 102. It is also possible that the timestamp is merely requested by local node 102 and cache manager 104 provides the timestamp.

[0033] After sending the timestamp at step 214, then at step 220, local node 102 invalidates the cache item of the named cache that is held by local node 102.

[0034] At step 222, after invalidating the cache item, local node 102 releases the write lock of the named cache.

[0035] At step 224, local node 102 broadcasts an invalidation message to all remote nodes 106, identifying the cache item of the named cache.

[0036] At step 226, each remote node 106 receives the broadcast invalidation message, identifying the cache item of the named cache.

[0037] At step 228, each remote node 106 acquires a local write lock of the named cache in their own remote node.

[0038] At step 230, each remote node 106 invalidates the cache item of the named cache in their own remote node.

[0039] At step 232, after invalidating the cache item of the named cache in their own remote node, each remote node 106 sends an acknowledgment (ack) message to local node 102.

[0040] At step 234, each remote node 106 releases the local write lock of the named cache in the remote node.

[0041] At step 236 and 238, after broadcasting the invalidation message at step 223, local node 102 waits for acknowledgment from each of the remote nodes 106.

[0042] It is possible that there is some delay in sending the acknowledgment (ack) from each remote node, or that the network connection to one or more of the remote nodes is lost and therefore the remote node does not receive the broadcast message, or is unable to send an acknowledgment. Therefore, although not illustrated, in one embodiment, a time-out timer may limit the length of time that local node 102 waits at steps 236 and 238 for all acknowledgments.

[0043] At step 240, after all acknowledgments have been received from remote nodes 106, local node 102 sends the update of the cache item in the named cache to master locking database 110.

[0044] At step 244, master locking database 110 receives the update and performs the update to cache item of the named cache in the master locking database.

[0045] At step 246, master locking database 110 commits the update of the cache item to the named cache of the master database.

[0046] At step 248, after confirmation of the commit at step 246, cache manager 104 releases the lock of the named cache (or global database).

[0047] As illustrated in FIG. 2, once the steps are complete, local node 102 and remote node(s) 106 have invalidated the cache item of the named cache in the remote nodes, but have not updated the cache item of the named cache in the local node or remote nodes. Therefore, when a request is made to either the local node 102 or the remote node(s) 106 for the cache item of the named cache, the local node or remote node generates a cache miss and must make a call to master database 110.

[0048] Referring now to FIG. 3, steps for dealing with a cache miss are illustrated. At step 302, the node (either the local node 102 or remote node 106 that generated the cache miss), identifies a cache miss of the cache item.

[0049] At step 304, the node (102 or 106) requests a read lock of the named cache (or global database).

[0050] At step 306, cache manager 104 receives the request, and read locks the named cache (or global database).

[0051] At step 308, cache manager 104 sends an indication to the node (102 or 106) that the named cache (or global database) is read locked.

[0052] At step 310, the node (102 or 106) receives an indication that the named cache (or global database) is read locked.

[0053] At step 312, the node (102 or 106) requests the cache item from master locking database 110.

[0054] At step 314, master locking database 110 receives the request for the cache item, and at step 316, master locking database 110 sends the cache item from the master locking database 110.

[0055] At step 318, the node (102 or 106) receives the cache item from master locking database 110.

[0056] At step 320, the node (102 or 106) sends a release of the read lock of the named cache (or global database) to cache manager 104.

[0057] At step 322, cache manager 104 receives the release of the read lock, and at step 324 releases the read lock of the named cache (or global database).

[0058] After completing the steps illustrated in FIG. 3, the node that experienced a cache miss of the cache item in the named cache has received an update of the cache item.

[0059] Referring now to FIG. 4, an embodiment of the invention includes cache manager 104, but does not include a separate master locking database 110. In this embodiment, the named cache is replicated and distributed among local node 102 and remote nodes 106.

[0060] Steps 202 through 238 are similar to or the same as the correspondingly numbered steps of FIG. 2.

[0061] At step 402, local node 102 updates the cache item of the named cache in the local node, and at step 404, local node 102 commits the update to the named cache

and ends.

[0062] Referring now to FIG. 5, which illustrates an embodiment for handing a cache when there is a cache manager but no separate master locking database. In this embodiment, the named cache is replicated and distributed among local node 102 and remote nodes 106. In the example illustrated in FIG. 5, the update of the cache item is made to local node 102, and remote node 106 experiences a cache miss.

[0063] Steps 302 through 310 and 320 through 324 are the same or similar to the similarly number steps of FIG. 3.

[0064] At step 502, remote node 106, which experienced the cache miss, requests the cache item from local node 102.

[0065] At step 504, local node 102 receives the request for the cache item, and at step 506, sends the requested cache item to remote node 106.

[0066] At step 508, remote node 106 receives the cache item from the local node and, although not illustrated in the figure, updates the cache item in the local cache of the remote node.

[0067] Referring now to FIG. 6, an embodiment of the invention helps to ensure that if a node becomes disconnected from the network, or misses a cache item invalidation, that failure will be detected and can be resolved. The steps illustrated in FIG. 6 are performed on a regular basis by each node.

[0068] At step 602, the node determines whether a previously set countdown timer of a predetermined length has expired, looping if the timer has not expired.

[0069] At step 604, when the countdown timer has expired, the node sends a request to cache manager 104 for a read lock of the named cache and the most current timestamp in the lock table.

[0070] At step 606, cache manager 104 receives the request for a read lock, and read locks the named cache.

[0071] At step 608, cache manager 104 gets the most current timestamp in the lock table.

[0072] At step 610, cache manager 104 sends the timestamp and an indication that the named cache is read locked to the requesting node.

[0073] At step 612, the node receives the indication that the named cache is read locked along with the timestamp.

[0074] At step 614, the node compares the timestamp received from the cache manager with the timestamp from the previous check.

[0075] At step 616, the node determines whether a timestamp is missing.

[0076] If, at step 616, the node determines that no timestamp is missing, then at step 618, the node stores the timestamp, received at step 612 as the timestamp of the last check.

[0077] At step 620, the node sends a message to the cache manager releasing the read lock of the named cache.

[0078] At step 622, in response to the message releasing the read lock of the named cache, cache manager 104 releases the read lock of the named cache and ends, or loops by re-starting the count-down timer that is monitored at step 602.

[0079] If at step 616, the node determines that a timestamp is missing, then at step 624, the node sends a request to master locking database 110 for an update of the named cache.

[0080] At steps 626, 628, master locking database 110 receives the request for an update of the named cache and sends the update to the node.

[0081] At step 630, the node receives the requested update and updates the named cache.

[0082] At steps 632 through 636, the node stores the timestamp and releases the read lock of the named cache, just as described with reference to steps 618 through 622.

[0083] Referring now to FIG. 7, an embodiment of the invention reorders some of the previously described steps and deletes some of the steps.

[0084] At step 702, local node 102 determines that a cache element in a named cache

requires modification. The need for modification may include the need for an update, or a data change.

[0085] At step 704, local node 102 sends the update of the cache item in the named cache to master locking database 110.

[0086] At step 706, master locking database 110 receives the update and performs the update to cache item of the named cache in the master locking database.

[0087] At step 708, master locking database 110 commits the update of the cache item to the named cache of the master database.

[0088] At step 710, local node 102 acquires a local write lock of the named cache. Although called a write lock, the write lock includes a read lock of the named cache.

[0089] At step 712, local node 102 sends a timestamp to cache manager 104 for use in an update of the lock table.

[0090] At step 714, cache manager 104 receives the timestamp, and at step 716 cache manager 104 updates the lock table with the timestamp from local node 102.

[0091] After sending the timestamp at step 712, then at step 718, local node 102 invalidates the cache item of the named cache that is held by local node 102.

[0092] At step 720, after invalidating the cache item, local node 102 releases the write lock of the named cache.

[0093] At step 722, local node 102 broadcasts an invalidation message to all remote nodes 106, identifying the cache item of the named cache.

[0094] At step 724, each remote node 106 receives the broadcast invalidation message, identifying the cache item of the named cache.

[0095] At step 726, each remote node 106 acquires a local write lock of the named cache in their own remote node.

[0096] At step 728, each remote node 106 invalidates the cache item of the named cache in their own remote node.

[0097] At step 730, after invalidating the cache item of the named cache in their own remote node, each remote node 106 sends an acknowledgment (ack) message to local node 102.

[0098] At step 732, each remote node 106 releases the local write lock of the named cache in the remote node.

[0099] At steps 734 and 736, after broadcasting the invalidation message at step 722, local node 102 waits for acknowledgment from each of the remote nodes 106.

[0100] The process is complete once all of the remote nodes have acknowledged the invalidation message.

[0101] Although illustrative embodiments have been described herein in detail, it should be noted and will be appreciated by those skilled in the art that numerous variations may be made within the scope of this invention without departing from the principle of this invention and without sacrificing its chief advantages.

[0102] Unless otherwise specifically stated, the terms and expressions have been used herein as terms of description and not terms of limitation. There is no intention to use the terms or expressions to exclude any equivalents of features shown and described or portions thereof and this invention should be defined in accordance with the claims that follow.